APPLICATION FOR

UNITED STATES LETTERS PATENT

SPECIFICATION

INVENTOR(S): Masanori KIRA

Title of the Invention: Storage Medium Storing Program Directing Computer to Control Optimization of Program, and Program Optimizing Apparatus

STORAGE MEDIUM STORING PROGRAM DIRECTING COMPUTER TO CONTROL OPTIMIZATION OF PROGRAM, AND PROGRAM OPTIMIZING APPARATUS

Background of the Invention

Field of the Invention

The present invention relates to the technology of optimizing a program, and more specifically to the technology of merging and deleting an unused data item in the program.

Description of the Related Art

Data items such as a variable, etc. are defined in a program in many cases. However, not all defined data items are not used in the program. In the following explanation, a data item defined but used in a program is referred to as an unused data item. Although a data item dictionary is generated during compilation, program analysis, etc., a larger memory requirement is required when unused data items are contained in the data item dictionary. Since an unused data item is not used in a program, a code for definition of the data item is not required. Therefore, a compiler is provided with the function of deleting a code for

definition of an unused data item in many cases so that memory sources can be effectively used.

The function of deleting a code for definition of an unused data item is described below by referring to a compiler. In addition to the above mentioned function, an actual compiler has the functions of removing a simple assignment, optimizing a constant calculation, etc. However, for simple explanation, it is assumed that a compiler does not perform optimization other than deletion of a code for definition of an unused data item.

FIG. 1 shows an example of a source program described in COBOL (COmmon Business Oriented Language). In the source program, since the variables IREC01, IDATA11, IDATA12, IDATA13, and IDATA14 defined in lines 130 through 170 are not referred to in the procedure division in and after line 300, they are unused data items. Similarly, the variables WDATA12, WDATA13, WREC02, WDATA21, WDATA22, and WDATA71 defined in lines 210, 220, and 240 through 270 are also unused data items.

When a code for definition of an unused data item is removed by the conventional technology from the source program shown in FIG. 1, the source

program shown in FIG. 2 can be obtained. In FIG. 2, the codes in lines 240 through 270 are removed.

FIG. 3 shows an example of a source program described in C. The source program shown in FIG. 3 has almost the same contents as the source program described in COBOL shown in FIG. 1. FIG. 4 shows a result of removing a record for definition of an unused data item from the source program shown in FIG. 3 by the conventional technology.

According to the compiler disclosed in the Japanese Patent Application Laid-open No. Hei 7-239788, it is determined whether or not a variable for which a data type is declared in the program has been used in the process description. If no use is determined, an error message is output to an output device, thereby notifying the user of the wasteful type declarlation in the program, and suppressing the generation of a variable for which a wasteful type declarlation is made. The compiler prevents a wasteful type declarlation from being included in a compiled program.

However, according to the above-mentioned conventional technology, the following unused data item cannot be removed to prevent the address from being shifted in the memory allotted to each data

item.

* data item forming at least a part of a record (a record is a data item having a hierarchical structure.)

Using the source program described in COBOL shown in FIGS. 1 and 2, the above-mentioned problems are practically described below. IREC01, IDATA11, IDATA12, IDATA13, IDATA14, WDATA12, and WDATA13 defined in the source program shown in FIG. 1 in lines 130 through 170, 210, and 220 are unused data items, but these unused data items are not removed in the source program shown in FIG. 2. Relating to the unused data item IREC01, IFILE01 is used in the SELECT statement in line 70. The unused data items IDATA11, IDATA12, IDATA13, and IDATA14 are data items configuring the IREC01. The unused data items WDATA12 and WDATA13 are data items configuring the record WREC01.

From the source program described in C shown in FIGS. 3 and 4, similar unused data items cannot be removed not only in COBOL but also in other languages.

However, the size of the memory area in a computer is limited. If unused data items in a source program which cannot be deleted are

processed as is, then the process may not continue because the available memory area possibly becomes insufficient during compilation, generation of a data item dictionary, etc.

To reduce the memory requirement when a source program is compiled and a data item dictionary is generated for the source program, it is desired that the smallest possible number of unused data items are contained in the source program.

Summary of the Invention

The present invention has been developed to solve the above mentioned problems, and aims at more effectively deleting the number of unused data items than the conventional technology, thereby optimizing a program to reduce the memory requirement during compilation and generation of a data item dictionary.

According to the first aspect of the present invention, the optimizing apparatus for optimizing a program includes: a data item extraction unit extracting data items from the program; a layout unit laying out the extracted data items on memory; an unused data item extraction unit extracting defined but unused data items from the data items;

a merge determination unit determining whether or not it is possible to merge a plurality of unused data items forming at least a part of a data item having a hierarchical structure into a new data item based in the layout; and a data item merge unit outputting the program in which the plurality of unused data items are merged into the new data item based on the determination result.

Conventionally, unused data items forming at least a part of a data item having a hierarchical structure cannot be deleted. However, the number of unused data items can be more effectively reduced than the conventional technology by merging a plurality of unused data items based on the layout on the memory with the above mentioned configuration. Therefore, a program can be optimized by reducing the memory requirement during compilation and generation of a data item dictionary.

The above mentioned unused data items to be merged can be laid out on adjacent areas on the memory.

When it is determined that data items can be merged, there are two merging methods. First, when the plurality of unused data items form another

data item having a hierarchical structure, and have the same hierarchical levels in the hierarchical structure, the merge determination unit determines that these data items can be merged. If a plurality of unused data items are a data item forming another data item having a hierarchical structure and a data item being the other data item, and the other data item having the hierarchical structure is formed by only one data item, then the merge determination unit determines that the plurality of unused data items can be merged.

Furthermore, the data item merge unit can delete a code for declaration of unused data item from the program, and add a code for declaration of a new data item. A program can be optimized by deleting an unnecessary definition code from the program.

Additionally, the data item merge unit can calculate a sum of the item lengths of the plurality of unused data items to be merged, and set the item length of a new data item based on the sum. Thus, the address of the data item on the memory before the merge can be prevented from being changed after the merge.

In merging the data items, the data type of a

new data item to be declared can be set in a plurality of methods. First, when the data types of the plurality of unused data items are all the same, the data item merge unit can set the data type of the new data item as the same data type as the plurality of unused data items to be merged. Thus, the data type of the data item before the merge can be prevented from being changed after the merge.

Furthermore, the data item merge unit can set the data type of the new data item as a data type of the smallest storage area. The data type of the smallest storage area size can be set as the data type of the new item, thereby easily setting the data type. An example of the case in which it is difficult to set a data type can be, for example, a case in which the data types of the plurality of unused data items to be merged are different. There is another case in which, to make the item length of a new data item equal to the sum of the item lengths of a plurality of unused data items before the merge, it is necessary to set a data type for a new data item different from the data type of the data item before the merge. For a practical example of the latter case, there is a case in which unused data items declared as numeric data for which the

precision of single-precision integer data, etc. is specified are to be merged.

In merging data items as described above, there are a plurality of methods of setting the item name of a new data item to be declared. For example, the data item merge unit can set the item name of the new data item as a blank. The data item merge unit can also set the item name of the new data item based on one of the unused data items to be merged. Additionally, the data item merge unit can set the item name of the new data item based on the specification of the user of the optimizing apparatus.

According to another aspect of the present invention, the method of optimizing a program, comprising: extracting a data item from the program; laying out the data item on the memory provided for the computer; extracting a defined but not unused data item from the extracted data item; determining whether or not a plurality of unused data items forming at least a part of the data items having hierarchical structures in the unused data items can be merged into a new data item based on the layout result; and outputting the program in which the plurality of data items are merged into

the new data item based on the above mentioned determination result. Since the program optimizing method can also obtain the operation and effect of the above mentioned optimizing apparatus, the above mentioned problems can be successfully solved.

Furthermore, the computer program used to direct a computer to execute the procedure in the block diagram program optimizing method can also solve the above mentioned problems by temporarily storing the computer program in the memory provided in the computer, and allowing the computer to read the computer program from the memory for execution.

Additionally, the above mentioned problems can be solved by allowing a computer to read the computer program from a computer-readable storage medium storing the computer program, loading the program, and then executing the program as described above.

Brief Description of the Drawings

The features and advantages of the present invention will be more clearly appreciated from the following description taken in conjunction with the accompanying drawings in which like elements are denoted by like reference numerals and in which:

FIG. 1 shows an example of a source program described in COBOL;

FIG. 2 shows a result of deleting codes for definition of unused data items from the source program shown in FIG. 1 by the conventional technology;

FIG. 3 shows an example of a source program described in C;

FIG. 4 shows a result of deleting codes for definition of unused data items from the source program shown in FIG. 3 by the conventional technology;

FIG. 5 shows the configuration of the optimizing apparatus;

FIG. 6 is a flowchart of the outline of the flow of the optimizing process;

FIG. 7 shows an example of a source program described in COBOL;

FIG. 8 is a view (1) of a data item dictionary of the source program shown in FIG. 7;

FIG. 9 is a view (2) of a data item dictionary of the source program shown in FIG. 7;

FIG. 10 is a view (3) of a data item dictionary of the source program shown in FIG. 7;

FIGS. 11A and 11B show notation methods of a

data item dictionary;

FIG. 12 is a view (1) of an example of the layout of a data item;

FIG. 13 is a view (2) of an example of the layout of a data item;

FIG. 14 is a view (3) of an example of the layout of a data item;

FIG. 15A shows an example of a layout of a data item;

FIG. 15B shows an example of a data item dictionary for FIG. 15A;

FIG. 15C shows an example of a layout after a back-and-forth direction merge;

FIG. 15D shows an example of a data item dictionary for the layout shown in FIG. 15C;

FIG. 16A shows an example of a layout of a data item;

FIG. 16B shows an example of a data item dictionary for FIG. 16A;

FIG. 16C shows an example of a layout after a set direction merge;

FIG. 16D shows an example of a data item dictionary for the layout shown in FIG. 16C;

FIG. 17 is a view (1) for explanation of a pointer included in an undetermined data item list;

FIG. 18 is a view (2) for explanation of a pointer included in an undetermined data item list;

FIG. 19 is a view for explanation of the notation method included in the undetermined data item list and the information included in the undetermined data item list shown in FIGS. 17 and 18;

FIG. 20 is a flowchart (1) of the procedure of determining a merge;

FIG. 21 is a flowchart (2) of the procedure of determining a merge;

FIG. 22 is a flowchart (3) of the procedure of determining a merge;

FIG. 23 is a flowchart (4) of the procedure of determining a merge;

FIG. 24 is a view (1) of the setting contents of the items A and B in each step of determining a merge on the source program shown in FIG. 7, and the contents of the undetermined data item list;

FIG. 25 is a view (2) of the setting contents of the items A and B in each step of determining a merge on the source program shown in FIG. 7, and the contents of the undetermined data item list;

FIG. 26 is a view (3) of the setting contents of the items A and B in each step of determining a

merge on the source program shown in FIG. 7, and the contents of the undetermined data item list;

FIG. 27 is a view (4) of the setting contents of the items A and B in each step of determining a merge on the source program shown in FIG. 7, and the contents of the undetermined data item list;

FIG. 28 is a view (5) of the setting contents of the items A and B in each step of determining a merge on the source program shown in FIG. 7, and the contents of the undetermined data item list;

FIG. 29 is a view (6) of the setting contents of the items A and B in each step of determining a merge on the source program shown in FIG. 7, and the contents of the undetermined data item list;

FIG. 30 is a view (7) of the setting contents of the items A and B in each step of determining a merge on the source program shown in FIG. 7, and the contents of the undetermined data item list;

FIG. 31 is a view (8) of the setting contents of the items A and B in each step of determining a merge on the source program shown in FIG. 7, and the contents of the undetermined data item list;

FIG. 32 shows a result of optimizing the source program shown in FIG. 7;

FIG. 33 shows a result of deleting

unnecessary definition codes from the optimized source program shown in FIG. 32;

FIG. 34 shows an example of the source program described in C;

FIG. 35 shows a result of optimizing the source program shown in FIG. 34;

FIG. 36 shows the configuration of a computer; and

FIG. 37 shows loading data and a program into a computer.

Description of the Preferred Embodiments

The embodiments of the present invention are described below by referring to the attached drawings. The same devices, etc. are assigned the same reference numerals, and double explanation is omitted.

FIG. 5 shows the configuration of an optimizing apparatus 1 according to the present invention. The optimizing apparatus 1 optimizes a source program such that the number of data items contained in the source program can be reduced by merging unused data items. As shown in FIG. 5, the optimizing apparatus 1 comprises a data item extraction unit 2, a data item dictionary

generation unit 3, a layout generation unit 4, an unused data item extraction unit 5, a merge determination unit 6, a determination result notification unit 7, a merge instruction reception unit 8, a data item merge unit 9, an input/output unit 10, a program storage unit 11, a data item dictionary storage unit 12, and an undetermined data item list storage unit 13.

The data item extraction unit 2 obtains a source program from the program storage unit 11, and extracts data items from the source program. The data item dictionary generation unit 3 generates a data item dictionary indicating the contents of the extracted data items and the relationship between data items, and writes it to the data item dictionary storage unit 12. The layout unit 4 arranges (lays out)the extracted data item in the memory (not shown in FIG. 5). The unused data item extraction unit 5 extracts unused data items in the extracted data items. The merge determination unit 6 determines based on the layout in the memory whether or not a group of a plurality of unused data items in the extracted unused data items can be merged into a data item. The determination result notification unit 7 notifies

the user of the optimizing apparatus 1 of the plurality of unused data items determined that they can be merged by the merge determination unit 6 through the input/output unit 10. The merge instruction reception unit 8 receives an instruction as to whether or not data items are to be merged through the input/output unit 10. The data item merge unit 9 merges data items according to an instruction from a user, and notifies the user of the merge result through the input/output unit 10. The input/output unit 10 is used in communicating information between a user and the optimizing apparatus 1.

The program storage unit 11 stores the source program input through the input/output unit 10, and the optimized source program. The data item dictionary storage unit 12 stores a data item dictionary generated by the data item dictionary generation unit 3. The undetermined data item list storage unit 13 stores an undetermined data item list. The undetermined data item list is generated by the merge determination unit 6 based on the extraction result and the layout result of unused data items, and is used to manage data items to be determined and determination results when it is

determined whether or not data item can be merged.

Then, by referring to FIGS. 6 through 27, the processes performed by the optimizing apparatus 1 are described below. First, the outline of the flow of the optimizing process of a program is described below by referring to FIG. 6.

As shown in FIG. 6, the optimizing apparatus 1 first retrieves the source program to be optimized from the program storage unit 11, analyzes the data flow in the source program, and collects the definitions of data items and reference information (S1). The analysis of a data flow is a common optimizing method of a program in a compiler, etc. The procedure of analyzing a data flow is described below.

1) The data item extraction unit 2 collects declaration information about data items for the translation text after the execution of a preprocess. An example of translation text can be a program obtained after extending a COPY statement of COBOL, an include statement of C, or C++, etc. A data item dictionary is explained later in detail.

2) The layout unit 4 calculates the relative displacement of the position in the layout of data items based on the item (outermost data item) which

is not a component of any data item in the data items forming a hierarchical structure, and collects overlapping information about data items. At this time, the layout unit 4 considers the correction of an interval between data items depending on the program language describing a source program. Interval corrections between data items can be, for example, an idle byte or bit in COBOL, a word boundary in C or C++, etc. The layout is described later in detail.

3) The data item dictionary generation unit 3 generates a data item dictionary according to declaration information, collects the definition/reference information about data items, and sets the definition/reference information in the data item dictionary. For example, when WDATA11 which is one of the data items forming the record WREC01 is referred to in the source program shown in FIG. 7, the data item dictionary generation unit 3 sets the definition/reference information for the WDATA1 and the record WREC01 in the data item dictionary.

Then, the unused data item extraction unit 5 extracts data items (that is, record) having a hierarchical structure in the extracted data items

(S2). For example, when the source program shown in FIG. 7 is optimized, the unused data item extraction unit 5 extracts the IREC01, WREC01, and WREC02 as a data item having a hierarchical structure.

When the unused data item extraction unit 5 cannot extract a data item having a hierarchical structure, that is, there is no data item having a hierarchical structure in the source program to be handled in the optimizing process (NO in S3), the process terminates.

When a data item having a hierarchical structure can be extracted (YES in S3), the unused data item extraction unit 5 extracts unused data items from data items having extracted hierarchical structures and data items (child items) forming data items according to the definition/reference information (not shown in FIG. 6).

The merge determination unit 6 generates an undetermined data item list based on the extraction result and the layout result of an unused data item, and determines whether or not some of the extracted unused data items can be merged into a smaller number of data items using the undetermined data item list (S4). The determination is described

later in detail.

Then, the determination result notification unit 7 determines whether or not the optimizing apparatus 1 is set such that a determination result can be output (S5). If it is set such that a determination result can be output (YES in S5), the determination result notification unit 7 notifies the user of a determination result by the merge determination unit 6 (S6). Described below is an example of the contents of the determination result supplied to the user.

1) An item name, an item length, and a data type of a data item to be merged

2) An item name, an item length, and a data type of a merged data item

3) optimized program

If it is not set such that a determination result can be output (NO in S5), then control is passed to S7.

If a merge instruction to merge data items which can be merged is set by the optimizing apparatus 1 in advance, or if the merge instruction reception unit 8 receives the user specification of data items to be merged after the determination result notification unit 7 notifies the user of a

determination result (S7), then the data item merge unit 9 merges unused data items based on the determination result contained in the undetermined data item list (S8), thereby terminating the process.

Described below are the data item dictionary and the layout generated in S1. In the explanation, an example of a source program described in COBOL shown in FIG. 7 is appropriately referred to. The program description language used in the following explanation is not used to limit the program description language applicable in the present invention.

First, the data item dictionary is described below by referring to FIGS. 8 through 11. FIGS. 8 through 10 show a data item dictionary relating to data items declared in the source program shown in FIG. 7. As shown in FIGS. 8 through 10, the data item dictionary indicates the relationship between data items declared in the source program using a pointer.

FIGS. 11A and 11B show the notation method adopted in FIGS. 8 through 10 to describe a data item dictionary. As shown in FIG. 11A, in the data item dictionary shown in FIGS. 8 through 10, each

data item is assigned five pointers. ϕ shown in FIG. 11B indicates that there are no corresponding values. The pointer indicates the following items.

1) A pointer pointing to the position (address) in which the parent item of the current data item is stored.

2) A pointer pointing to the position in which a data item positioned before the current data item is stored in assigning an address in the memory.

3) A pointer pointing to the position in which a leading data item in the memory in the data items which are child items of the current data item is stored.

4) A pointer pointing to the position in which a data item positioned after the current data item is stored in assigning an address in the memory.

5) A pointer pointing to the position in which other data items having no parent items are stored.

A child item refers to a data item forming a data items having a hierarchical structure. A parent item refers to a data item having the hierarchical structure formed by one or more other data items. For example, in the data item dictionary shown in FIGS. 8 through 10, the data items IDATA11, IDATA12, IDATA13, and IDATA14

forming the data item IREC01 are child items of the data item IREC01. Inversely, the data item IREC01 is a parent item of the data items IDATA11, IDATA12, IDATA13, and IDATA14.

Furthermore, the pointer pointing to the position in which other data items having no parent items described in 5) above are stored is required in performing the optimizing process. However, a method other than the above mentioned method using the pointers can also be adopted. For example, as shown in FIG. 11B, all data items can be connected in series regardless of the hierarchical relationship among data items.

The layout of data items is described below by referring to FIGS. 12 through 14. First, FIG. 12 shows the layout of the data items declared by the code described in lines 130 through 170 of the source program described in COBOL shown in FIG. 7. In the layout shown in FIG. 12, the interval between data items is not corrected.

FIG. 13 shows the layout of the data items declared by the following COBOL code. The record DATA11 is formed by the data items DATA21 and DATA22. The data item DATA21 is formed by the data items DATA31, DATA32, and DATA33. The data item

DATA22 is formed by the data item DATA34. In the layout shown in FIG. 13, the layout unit 4 inserts an idle bit or an idle byte between the data items DATA 31 and DATA32, and after the data item DATA34.

```
01    DATA11.
 02    DATA21.
   03      DATA31 PIC X(5).
   03      DATA32 PICS9(4) BINARY SYNCHRONIZED.
   03      DATA33 PICS9(4) BINARY SYNCHRONIZED.
 02    DATA22.
   03      DATA34 PIC 1(4) BIT    SYNCHRONIZED.
```

FIG. 14 shows the layout of the data items declared by the following code in C. As shown by the following code, the record irec01 is formed by the data items idata11 and idata12. In the layout shown in FIG. 14, a blank byte called padding is provided for alignment between the variable idata11 for storing a character and the variable idata12 for storing an integer. In some program environments, it is necessary that a leading address in the memory assigned to a data item can be divided by a given integer depending on the data type. This is called "alignment". The system of

alignment depends on a process system.

```
struct {
  char   idata11[3];
 int    idata12   ;
} irec01;
```

Then, the outline of the merge determining process in S4 shown in FIG. 6 is described. The above mentioned merge determining process is performed as follows.

1) First, the merge determination unit 6 extracts a data item having a hierarchical structure, that is, a record, from a source program. In the case of the source program shown in FIG. 7, the following data items are extracted.

-IREC01, WREC01, WREC02

2) Then, the merge determination unit 6 extracts an unused data item according to the definition/reference information about the data item for the data item extracted in 1) above. In the case shown in FIG. 7, the following data items are extracted.

-IREC01, IDATA11, IDATA12, IDATA13, IDATA14

-WDATA12, WDATA13

-WREC02, WDATA21, WDATA22

3) The consecutiveness of data items is determined for the unused data items extracted in 2) above.

The determination in 3) above is performed by selecting the two data items (hereinafter referred to as items A and B) from the unused data items (hereinafter referred to as an undetermined data item) which have not been determined. There are the following two types of data item merging methods. One is a back-and-forth direction merge, and the other is a set direction merge. There are also two types of determination conditions corresponding to the respective merging methods. The two types of merges are described below.

a) Back-and-forth direction merge

A back-and-forth direction merge refers to a merge of a plurality of data items at the same hierarchical level in the hierarchical structure of a data item. The merging conditions are described below.

- Having the same parent item.
- Having no child items
- The "relative displacement of item A in the layout + item length of item A" matches the "relative displacement of item B in the layout".

When they match, the items A and B are adjacent (consecutive) in the layout. When an idle byte or an idle bit, etc. is used for correction, there are the conditions that the "relative displacement of item A in the layout + item length of item A + correction value" matches the "relative displacement of item B in the layout".

In the back-and-forth direction merge, a merged data item is represented as follows.

- A merged data item has no name (no data item name), or has a data item name generated by the data item merge unit 9.
- To make the item length of a merged data item match the item length of the data item before the merge, the sum of the item lengths of a plurality of data items to be merged is defined as a data item length of the merged data item. When the layout is corrected by the optimizing apparatus 1 using an idle byte, an idle bit, or padding depending on the program language, the correction is to be considered.
- The data type of a merged data item is to be the same as the data type before the merge, or the type of data having the smallest storage area size. In the latter case, although the data types of a

plurality of data items to be merged do not match, the merging process can be performed.

The back-and-forth direction merge is described below by referring to FIGS. 15A to 15D.

The layout of the data item declared by the following code and the data item dictionary are shown in FIGS. 15A and 15B.

```
01  A1.
  02   B1    PIC X(10).
  02   B2    PIC X(10).
```

When the data items B1 and B2 are unused data items, the above mentioned code is represented as follows by performing the back-and-forth direction merge on the data items B1 and B2. The layout and the data item dictionary shown in FIGS. 15A and 15B are respectively shown in FIGS. 15C and 15D. As shown in FIGS. 15C and 15D, it is proved that the number of data items can be reduced by the back-and-forth direction merge.

```
01    A1.
  02    FILLER    PIC X(20).
```

b) Set direction merge

A set direction merge refers to a merge of a plurality of data items having different hierarchical structures (inclusion relation).

The following merging conditions are set.

- The item A is a parent item of the item B.
- The "item length of the item A" matches the "item length of the item B".

In the set direction merge, a merged data item is represented as follows.

- A merged data item has no name (no data item name), or has a data item name generated by the data item merge unit 9.
- The item length of a merged data item is equal to the item length of the item B.
- The data type of a merged data item is the data type of the item B.

The set direction merge is described below by referring to FIGS. 16A to 16D.

The layout of data items declared by the following code and the data item dictionary are represented as shown in FIGS. 16A and 16B.

```
01  A1.
  02  FILLER    PIC X(20).
```

When the data items A1 and FILLER are unused data items, the above mentioned code is represented as follows by performing a set direction merge on the data items A1 and FILLER. Furthermore, the layout and the data item dictionary shown in FIGS. 16A and 16B are changed as represented as shown in FIGS. 16C and 16D. As shown in FIGS. 16C, it is proved that the number of data items has been reduced.

```
01   FILLER   PIC X(20)
```

There can be the case in which an unused data item has a hierarchical structure of two or more hierarchical levels. In this case, a process may not be sufficiently performed in one optimizing process. Therefore, it is necessary to repeatedly perform the above mentioned determining process. When the process is repeatedly performed, the optimizing process is performed on all extracted unused data items on the following conditions.

- A data item having deeper levels is merged by priority.
- After performing the set direction merge, the

back-and-forth direction merge is performed, thereby recursively making determination.

The processing of data items having two or more hierarchical levels is described below by repeating the determining process by referring to actual code.

First, the following initial code is assumed. For explanation, all data items declared by the code are assumed to be unused data items.

```
01  A1.
    02  B1.      PIC X(20).
    02  B2.
      03  C1     PIC X(10).
      03  C2     PIC X(10).
    02  B3.      PIC X(20).
```

In the first optimizing process, the data items C1 and C2 having the deepest hierarchical levels are merged. Therefore, the above mentioned initial code is represented as follows.

```
01  A1.
    02  B1.      PIC X(20).
    02  B2.
```

```
        03  FILLER PIC X(20).      *>
      02  B3.      PIC X(20).
( *> first: back-and -forth direction merge)
```

Furthermore, when the second optimizing process is performed on the code after the first optimizing process, the data item B2 and the FILLER item are merged. As a result, the following code after the second optimizing process is obtained.

```
    01  A1.
      02  B1.      PIC X(20).
      02  FILLER   PIC X(20).      *>
      02  B3.      PIC X(20).
( *> second: set direction merge)
```

Furthermore, when the third optimizing process is performed on the code after the second optimizing process, the data item B1 and the FILLER item are merged. As a result, a code after the following third optimizing process can be obtained.

```
    01  A1.
      02  FILLER   PIC X(40).      *>
      02  B3.      PIC X(20).
```

(*> third: back-and forth direction merge)

Furthermore, when the fourth optimizing process is performed on the code after the third optimizing process, the data item B3 and the FILLER item are merged. As a result, a code after the following fourth optimizing process can be obtained.

```
01  A1.
    02  FILLER   PIC X(60).      *>
( *> fourth: back-and-forth direction merge)
```

Finally, when the fifth optimizing process is performed on the code after the fourth optimizing process, the data item A1 and the FILLER item are merged. As a result, a code after the following fifth optimizing process can be obtained.

```
01  FILLER    PIC X(60).      *>
( *>  fifth: set direction merge)
```

Since the code after the fifth optimizing process cannot be merged any more, the process terminates.

Described below is a merge determination by

the merge determination unit 6 in detail. The merge determination is made on an unused data item extracted by the unused data item extraction unit 5 immediately before S4 shown in FIG. 6. In a merge determination, the merge determination unit 6 uses the undetermined data item list for management of an undetermined data item and a determination result. The undetermined data item list includes the information about an undetermined data item and a determination result, a pointer to a data item dictionary, and a pointer to a data item to be determined next.

FIGS. 17 and 18 show a pointer to the undetermined data item list relating to the source program shown in FIG. 7. In FIGS. 17 and 18, ϕ indicates that there are no corresponding values. FIG. 19 shows the notation of a pointer used in FIGS. 17 and 18, and the detailed information other than the pointer contained in the undetermined data item list. As shown in FIG. 19, the information contained in the undetermined data item list are listed below.

1) Information about whether or not a data item is undetermined (hereinafter referred to as status information).

2) Information about whether or not merge-enabled or deletion-enabled (hereinafter referred to as determination result information)

3) Item length of a merged data item (hereinafter referred to as item length information)

4) Data type of a merged data item (hereinafter referred to as type information)

5) Item name of a data item set as items A and B

In FIGS. 17 and 18, the data item pointed to as a data item to be determined is a data item determined as an unused data item by the unused data item extraction unit 5 immediately before S4 shown in FIG. 6. The data item to be determined for a merge is selected based on the pointer contained in the undetermined data item list. The determination result contained in the undetermined data item list is referred to by the data item merge unit 9 when data item are merged.

The flow of the process performed in determining a merge is described below by referring to the flowchart shown in FIGS. 20 and 21.

First in the merge determination, the merge determination unit 6 defines the status information about the data items in the undetermined data item list as "undetermined", thereby entering an unset

state of the items A and B to be handled in the determining process (S11 and S12). Then, the merge determination unit 6 determines whether or not there is an undetermined data item by referring to the status information in the undetermined data item list (S13). If the merge determination is performed for all data items (NO in S13), then control is passed to S42 (described later). If there is an undetermined data item, then the merge determination unit 6 retrieves a data item from the undetermined data items based on the pointer contained in the undetermined data item list (S14). For example, when the source program shown in FIG. 7 is optimized, the data item IREC01 is first retrieved.

Then, the merge determination unit 6 determines whether or not the item A has already been set according to the status information in the undetermined data item list (S15). If the item A has not been set yet (YES in S15), then the merge determination unit 6 sets the status information about the retrieved data item in the undetermined data item list as "determined", thereby setting the data item as an item A (S16). After the setting in S16, control is returned to S13. If the item A has

already been set, the merge determination unit 6 sets the retrieved data item as an item B (S17). Since the setting in S17 is similar to the setting in S16, the explanation is omitted here.

After the items A and B are set, the merge determination unit 6 determines based on the data item dictionary whether or not the item A is a parent item of the item B (S18). If the item A is the parent item of the item B (YES in S18), then the merge determination unit 6 returns to "undetermined" the status information about the data item set as the item B in the undetermined data item list, thereby returning the item B to the unset status. Furthermore, the merge determination unit 6 performs a recursive call using as an argument the pointer in the undetermined data item list pointing to the item to be determined next. After the recovery from the recursive call, control is passed to S20 shown in FIG. 21. If the item A is not the parent item of the item B (NO in S18), then the merge determination unit 6 enters S27 shown in FIG. 18.

In S20, after the recovery from the recursive call, the merge determination unit 6 determines whether or not the item A has a child item other

than the item B according to the undetermined data item list and the data item dictionary. If a child item is set as "deletion-enabled" in the determination result information in the undetermined data item list although the child item is contained in the data item dictionary, then the merge determination unit 6 processes the child item as a non-existing child item.

If the item A has no child items other than the item B (YES in S20), control is passed to S21, and the set direction merge is performed on the items A and B.

In S21, the merge determination unit 6 sets the data type of the item B in the type information about the item A in the undetermined data item list. Additionally, the merge determination unit 6 sets "merge-enabled" in the determination result information about the item A in the undetermined data item list (S22), and sets "deletion-enabled" in the determination result information about the item B (S23). The merge determination unit 6 further sets the item length of the item B in the item length information about the item A. Then, the merge determination unit 6 defines the item B as unset (S24), thereby control is returned to S13.

If the item A has a child item other than the item B (NO in S20), then the merge determination unit 6 defines the items A and B as unset (S25 and S26), thereby returning control to S13 shown in FIG. 20.

If the item A is not the parent item of the item B in S18 described above (NO in S18), then the merge determination unit 6 determines in S27 whether or not the items A and B have the same parent item. If the items A and B have the same parent item (YES in S27), then control is passed to S28. Otherwise (NO in S27), control is passed to S38 shown in FIG. 23.

In S28, the merge determination unit 6 determines whether or not the item B has a child item based on the undetermined data item list and the data item dictionary (S28). The determination in S28 is similar to the determination in S20.

When the item B has a child item (YES in S28), a recursive call is made using as an argument the pointer pointing to the data item to be determined next in the undetermined data item list (S29), thereby passing control to S30. If the item B has no child items (NO in S28), then control is passed to S30 without performing the process in S29. If

the item B has a child item (NO in S30), then control is passed to S37.

In S30, the merge determination unit 6 determines whether or not the item B has a child item. If the item B has no child items (YES in S30), then the merge determination unit 6 determines based on the layout generated by the layout unit 4 whether or not the area storing the item A and the area storing the item B are consecutive areas (S31). If the area storing the item A and the area storing the item B are consecutive areas, then control is passed to S32, and the back-and-forth direction merge is performed on the items A and B. Otherwise, control is passed to S37.

In S32, the merge determination unit 6 computes the item length of a merged data item by adding the item length of the item B to the item length of the item A based on the data item length of the data item shown in the layout. Then, the merge determination unit 6 sets the computed item length in the item length information about the item A. When any correction such as an idle bit, an idle byte, padding, etc. is made during layout, the correction value is added to the sum of the item length of the item A and the item length of the

item B, thereby computing the length of the merged item.

Then, the merge determination unit 6 sets the type information about the item A in the undetermined data item list as the data type having the smallest storage area size (S33). Thus, the data type of the merged data item is determined. When the data type of the item A is the same as the data type of the item B, the data type can be kept unchanged.

Furthermore, the merge determination unit 6 sets the determination result information in the undetermined data item list as "merge-enabled" (S34), and sets the determination result information of the item B as "deletion-enabled" (S35). Then, the merge determination unit 6 defines the item B as unset (S24), and control is returned to S13 shown in FIG. 20

On the other hand, if the determination is NO in S30 or S31, the merge determination unit 6 sets as the item A again the data item currently set as the item B, and defines the item B as unset. Then, control is returned to S13 shown in FIG. 20.

When the determination in S27 is NO, the merge determination unit 6 determines whether or not a

recursive call has been made (S38). If a recursive call has not been made (YES in S38), then the merge determination unit 6 sets as the item A the data item currently set as the item B (S39), and also sets the item B as unset (S40). Then, control is returned to step S13. If a recursive call is made (NO in S38), the merge determination unit 6 sets as "undetermined" the status information about the data item currently set as the item B, and control is returned to the caller of the recursive call (S41). Furthermore, control is returned to the caller, that is, the caller of the recursive call or the main flow (S42), thereby terminating the determining process.

FIGS. 24 through 31 show the setting contents of the items A and B, and the contents of the undetermined data item list when a merge process is performed for the source program shown in FIG. 7. FIGS. 24 through 27 show the level number for identification of a level, a step number for identification of the procedure of the determining process, an item name of the data item stored in the primary storage area retrieved by the merge determination unit 6, an item name of the data item set as an item A, an item name of the data item set

as an item B, the status information and the determination result information in the undetermined data item list. In FIGS. 24 through 31, ϕ indicates "no corresponding data". The numeric character enclosed by the parentheses after the step number indicates the presence/absence of a recursive call. If the numeric character enclosed by the parentheses is 0, then there is no recursive call. If it is 1, then a recursive call has been made (it is necessary to return to the step of the caller).

The notation of the undetermined data item list in FIGS. 24 through 31 is shown below. The description of the information other than the status information and the determination result information is omitted.

data item name (status information, determination result information)

- status information: undetermined
  determined
- determination result information:
  merge-enabled
  deletion-enabled

For example, IREC01 (determined, merge-enabled) indicates that the data item IREC01 is

"determined", and the determination result is "merge-enabled".

The merge determination is practically explained below by referring to FIGS. 24 through 31.

For example, in S00 at the stage 1 (before performing the determining process), the undetermined data items can be:

IREC01, IDATA11, IDATA12, IDATA13, IDATA14, WDATA12, WDATA13, WREC02, WDATA21, WDATA22

Then, at the stages 2 and 3, the steps S11 and S12 are performed respectively to define the items A and B as unset. At the stage 4, step S14 is performed to retrieve the data item IREC01 and store it in a temporary storage area. At the stage 5, step S15 is performed to set the status information of IREC01 from "undetermined" to "determined". Then, at the stages 6 and 7, the data item IDATA11 is retrieved from the undetermined data item list and set as an item B, a recursive call is made, and control is passed to S0 (during recursive call).

At the stages 8 through 14, the items A and B are set. As shown at the stage 14 shown in FIG. 25, the data items set as the items A and B are IDATA11 and IDATA12 respectively. As clearly shown by the

data item dictionary in FIG. 17, IDATA11 and IDATA12 are not in the set relationship, and IDATA12 has no child items. Therefore, the processes in S27 through S36 are performed at the stages 15 through 17. As a result, at the stage 17 as shown in FIG. 26, the status information and the determination result information about the IDATA11 are set as "determined" and "merge-enabled" respectively, and the status information and the determination result information about the IDATA12 are set as "determined" and "deletion-enabled" respectively. That is, IDATA11 and IDATA12 are set for the back-and-forth direction merge. Furthermore, the item B is defined as unset.

At the stages 18 through 21 shown in FIG. 22 after the stage 17 shown in FIG. 36, the IDATA11 and IDATA12 are set for the back-and-forth direction merge by performing S13 through S26 as described above. Furthermore, at the stages 22 through 25 shown in FIG. 27, the IDATA11 and IDATA14 are set as merge-enabled for the back-and-forth direction merge as described above. Then, by performing S13 through S17 from the stages 26 and 27 shown in FIG. 27 after the stage 25 shown in FIG. 27, WDATA12 is set as an item B. Since IDATA11 is

not a parent item of WDATA12 (NO in the determination in S18), and the parent item of IDATA11 is not the same as the parent item of WDATA12 (NO in the determination in S27), S41 is performed at the stage 28 shown in FIG. 27, thereby control is returned to the caller S19. At this time, the items A and B are IREC01 and IDATA11 set during recursive call.

IREC01 is the parent item of IDATA11, and has no child items other than IDATA11 as a result of the back-and-forth direction merge as described above. Therefore, at the stages 29 through 31 shown in FIG. 28, S22 through S24 are performed. As a result, as shown in FIG. 28, the status information and the determination result information of IREC01 are respectively set as "determined" and "merge-enabled", and the status information and the determination result information of IDATA11 are respectively set as "determined" and "deletion-enabled" at the stage 31. That is, it is determined that the back-and-forth direction merge can be performed for IREC01 and IDATA11. Furthermore, the item B is defined as unset. By the merge determination so far, it is determined that the data items IREC01, IDATA11, IDATA12, IDATA13 can be

merged to be one data item.

By performing S13 through S17 at the stages 32 and 33 shown in FIG. 28 after the stage 31 shown in FIG. 28, IREC01 is set as an item A, and WDATA12 is set as an item B. However, since IREC01 is not the parent item of WDATA12, and the parent item of IREC01 is not the same as the parent item of WDATA12, it is determined that these items cannot be merged. As a result, by performing S39 and S40 at the stages 34 and 35 shown in FIG. 28, WDATA12 is set again as an item A, and an item B is unset, thereby returning control to S13. Afterwards, as described above, it is determined that the back-and-forth direction merge can be performed on WDATA12 and WDATA13 in the four data items forming the data item WREC01 from the stage 36 shown in FIG. 29 to the stage 40 shown in FIG. 30. Furthermore, from the stage 41 shown in FIG. 30 to the stage 59 shown in FIG. 31, it is determined that the back-and-forth direction merge and the set direction merge can be performed on the data item WREC02 and the two data items forming the data item.

The determination result stored in the undetermined data item list is transmitted to the user by the determination result notification unit

7. When the merge instruction reception unit 8 receives a merge instruction from a user, the data item merge unit 9 merges the data items based on the determination result, thereby optimizing the source program.

FIG. 32 shows a program obtained by optimizing the source program shown in FIG. 7. In lines 130 through 170 of the source program shown in FIG. 7, the following five data items are declared. As clearly indicated by the description of the procedure division of the source program shown in FIG. 7, these data items are unused data items.

- IREC01
- DATA11 (data type: character data, item length: 20)
- DATA12 (data type: character data, item length: 20)
- DATA13 (data type: character data, item length: 20)
- DATA14 (data type: character data, item length: 20)

As a result of the merge, it is indicated in line 130 shown in FIG. 32 that these data items are replaced with one FILLER item having the item length of 80, whose data type is character data.

That is, since five data items are merged into one data item, the number of items is reduced.

Similarly, the two data items WDATA12 and WDATA13 which are declaired in lines 210 and 220 in the source program shown in FIG. 7, and are numeric data each having the item length of 2 are merged into one FILLER item of numeric data having the item length of 4 as a result of the optimizing process. Furthermore, the three data items declared in lines 240 through 260 of the source program shown in FIG. 7 are merged into one FILLER item.

In this example, when a plurality of data items to be merged are the same in data type, the data type of the merged data item is the same as the data items to be merged. However, the data type of the merged data item can be the data type for the smallest storage area. In any case, since the item length of the merged FILLER item is set for the length of the merged data item, the address in the memory of the merged data item is not changed. Therefore, a plurality of data items which are part of a record do not cause the problem that the address in the memory of the data items is shifted.

The above mentioned optimizing apparatus 1 can further comprise an unused data item deletion unit

(not shown in the attached drawings) for realizing the function of deleting the code for definition of an unused data item. Since the function of deleting the code for definition of an unused data item has been conventionally assigned to a compiler, etc., the detailed explanation is omitted here.

In the optimizing apparatus 1 comprising the unused data item deletion unit, the unused data item deletion unit deletes the code for declaration of an unused data item from the source program to be manipulated in the optimizing process before merge determination of data items, thereby performing a preprocess for merge determination, and storing the preprocessed source program in the program storage unit 11. By performing the preprocess, the number of unused data items to be processed in the merge determination can be reduced.

Then, as described above, the data item extraction unit 2, the data item dictionary generation unit 3, the layout unit 4, the unused data item extraction unit 5, the merge determination unit 6, and the data item merge unit 9 merge the unused data items in the preprocessed source program retrieved from the program storage unit 11. Furthermore, the unused data item deletion

unit again deletes the code for declaration of an unused data item from the source program for merging the unused data items.

FIG. 33 shows the result of deleting an unnecessary definition code after merging the unused data items from the source program shown in FIG. 7. As shown in FIG. 33, the optimizing apparatus 1 has a smaller number of unused data items in the source program as compared with the deletion result of the unnecessary definition code in the conventional technology shown in FIG. 2.

FIG. 34 shows an example of a source program described in C language. The source program shown in FIG. 34 corresponds to the source program described in COBOL shown in FIG. 7. As the source program described in COBOL, the unused data items in the source program described in C language can be merged. If the source program shown in FIG. 34 can be optimized by merging the unused data items, the source program as shown in FIG. 35 is obtained. Since there are no data items corresponding to the FILLER item of COBOL in C language, the item name of the merged data item is the same as the item name of the data item having the first address on the layout in the data items to be merged in FIGS.

34 and 35. The data type "short" (single precision integer data) is assumed to be 2 bytes.

As shown in FIGS. 34 and 35, the number of unused data items in a source program described in C language can be reduced as in the source program described in COBOL language. That is, the optimizing apparatus 1 can perform the optimizing process independent of the language.

The optimizing apparatus 1 described above can be configured by a computer. The configuration of the computer is described below by referring to FIG. 36.

As shown in FIG. 36, a computer 20 comprises a CPU 21, memory 22, an input device 23, an output device 24, an external storage device 25, a medium drive device 26, and a network connection device 27, and these components are interconnected through a bus 28.

The memory 22 contains, for example, ROM (read only memory), RAM (random access memory), etc., and stores a program and data used in processing. A source program read from each storage unit 15, 16, or 17 is temporarily stored in the memory 22. The data item extracted by the data item extraction unit 2 is laid out by the layout unit 4 on the

memory 22. The data item extraction unit 2, the data item dictionary generation unit 3, the layout unit 4, the unused data item extraction unit 5, the merge determination unit 6, the determination result notification unit 7, the merge instruction reception unit 8, and the data item merge unit 9 forming the optimizing apparatus 1 are stored as a program in the specific program code segment of the memory 22 of the computer 20. Furthermore, the unused data item deletion unit can also be designed to be stored as a program in the program code segment of the memory 22 of the computer 20. The CPU 21 performs a necessary process by performing the above mentioned program using the memory 22.

The input device 23 can be, for example, a keyboard, a pointing device, a touch panel, etc., and is used in inputting a user instruction and information. The output device 24 can be, for example, a display, a printer, etc., and is used in outputting an inquiry to a user of a computer, a process result, etc. The input device 23 and the output device 24 correspond to the input/output unit 10 shown in FIG. 5.

The external storage device 25 can be, for example, a magnetic disk device, an optical disk

device, a magneto-optical disk device, etc. The external storage device 25 realizes a program storage unit 11, a data item dictionary storage unit 12, and an undetermined data item list storage unit 13. Furthermore, the above mentioned program can be stored in the external storage device 25 of the computer 20, and can be loaded into the memory 22 for use as necessary.

The medium drive device 26 drives a portable storage medium 29 and accesses the stored contents. The portable storage medium 29 can be any computer-readable storage medium such as a memory card, a memory stick, a flexible disk, CD-ROM (compact disc read only memory), an optical disk, a magneto-optical disk, a DVD (digital versatile disk), etc. The above mentioned program can be stored in the portable storage medium 29, and loaded into the memory 22 of the computer 20 for use as necessary.

The network connection device 27 communicates with an external device through any network (line) such as a LAN, a WAN, etc. for data conversion required in communications. Furthermore, the above mentioned program is received from an external device as necessary, and loaded into the memory 22 of the computer 20 for use as necessary.

FIG. 37 shows the loading of a program into the computer shown in FIG. 36.

A program used to direct a computer to realize the functions corresponding to the optimizing apparatus 1 can be input directly from the input device 23 of the computer, but can also be loaded into the computer as follows. For example, the computer-readable portable storage medium 29 stores the above mentioned program in advance. Then, as shown in FIG. 37, the program is read by the computer from the portable storage medium 29, and temporarily stored in the memory 22 of the computer or the external storage device 25. To allow the computer to perform the optimizing process, the stored program is read by the CPU 21 of the computer for execution.

Furthermore, a program can be downloaded to the computer from the DB of a program (data) provider 30 through a communications line (network) 31. In this case, for example, the transmitting computer of the program (data) provider 30 converts program data representing a program into a program data signal, the converted program data signal is modulated by a modem to obtain a transmission signal, and the obtained transmission signal is

output to the communications line 31 (transmission medium). In the computer for receiving a program, the transmission signal received by the modem is demodulated, thereby obtaining a program data signal, and the obtained program data signal is converted to obtain program data. If the communications line 31 (transmission medium) connecting the transmitting computer to the receiving computer is a digital line, a program data signal can be communicated.

Described above are the embodiments of the present invention, but the present invention is not limited to the above mentioned embodiments, but can be any variation. The above mentioned optimizing apparatus 1 can be applied to a compiler, etc. Thus, the memory requirement for a compiler can be reduced, thereby efficiently performing a compiling process. Furthermore, the optimizing apparatus 1 can be used in generating a data item dictionary.

As described above in detail, according to the present invention, the number of unused data items can be reduced by merging a plurality of unused data items forming a data item having a hierarchical structure into one data item based on the hierarchical level in the hierarchical

structure and the arrangement of the data items in the memory. Then, a program can be optimized such that the necessary memory requirement during compilation or generation of a data item dictionary can be reduced.

While the invention has been described with reference to the preferred embodiments thereof, various modifications and changes may be made to those skilled in the art without departing from the true spirit and scope of the invention as defined by the claims thereof.